Đọc dữ liệu data.txt

```
import pandas as pd
import numpy as np
from apyori import apriori
data= pd.read_csv('data.txt', header=None)
```

Chuyển dữ liệu về dạng ngang

Chuyển dữ liệu về dạng ngang

```
records = []
for i in range(len(data)):
    transaction = [str(item) for item in data.loc[i] if pd.notna(item)]
    records.append(transaction)
records
```

```
[['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk'],
 ['Bread', 'Butter', 'Milk'],
 ['Chips', 'Butter', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Milk'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Apple'],
 ['Wine', 'Chips', 'Milk'],
 ['Wine', 'Bread', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk'],
 ['Chips', 'Bread', 'Butter', 'Apple'],
 ['Wine', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk'],
 ['Wine', 'Bread', 'Milk', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Chips', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Milk', 'Apple'],
 ['Chips', 'Bread', 'Milk']]
```

Trong đó pd.notna nhằm mục đích kiểm tra giá trị có bị ‘ NaN’ hay không

Áp dụng apriori với minsup =3

```
association_rules = apriori(records, min_support=0.5, min_confidence=0.7, min_lift=1.1, min_length=3)
results = list(association_rules)
results
```

Điều chỉnh lại hàm để xuất ra kết quả

```
import pandas as pd

def convert_to_dataframe(results):
    data = []
    for rule in results:
        left_hand_side = ', '.join(list(rule[2][0][0]))
        right_hand_side = ', '.join(list(rule[2][0][1]))
        support = rule[1]
        confidence = rule[2][0][2]
        lift = rule[2][0][3]
        data.append([left_hand_side, right_hand_side, support, confidence, lift])

    df = pd.DataFrame(data, columns=['Left_Hand_Side', 'Right_Hand_Side', 'Support', 'Confidence', 'Lift'])
    return df

df = convert_to_dataframe(results)
print(df)
```

	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
0	Apple	Bread	0.545455	0.800000	1.100000
1	Bread	Butter	0.590909	0.812500	1.191667
2	Bread	Wine	0.590909	0.812500	1.117188
3	Butter	Milk	0.590909	0.866667	1.121569
4	Milk	Wine	0.636364	0.823529	1.132353
5	Butter	Milk, Bread	0.500000	0.733333	1.241026
6	Milk, Bread	Wine	0.500000	0.846154	1.163462

Tại dòng thứ 5,6 tồn tại liên kết 3 nhân tố

Tự code

Đếm số lượng các items

```
def count_item(df_items):
    count_ind_item = {}
    for row in df_items:
        for i in range(len(row)):
            item = row[i]
            if item != 'NaN':
                if item in count_ind_item.keys():
                    count_ind_item[item] += 1
                else:
                    count_ind_item[item] = 1

    data = pd.DataFrame()
    data['item_sets'] = count_ind_item.keys()
    data['supp_count'] = count_ind_item.values()
    return data
```

Đếm số lần xuất hiện của các items

```
x=count_item(records)
x
```

	item_sets	supp_count
0	Wine	17
1	Chips	15
2	Bread	17
3	Butter	16
4	Milk	18
5	Apple	16

```
def frequently_items(list_items):
    itemsets = []
    for i, entry in enumerate(list_items):
        proceding_items = list_items[i+1:]
        itemsets.extend([(entry, item) for item in proceding_items if isinstance(item, str) and entry != item])
        itemsets.extend([entry + item[1:] for item in proceding_items if not isinstance(item, str) and entry[:-1] == item[:-1]])

    if len(itemsets) == 0:
        return None

    return itemsets
```

Tạo các cặp item với nhau

```
[('Wine', 'Chips'),
 ('Wine', 'Bread'),
 ('Wine', 'Butter'),
 ('Wine', 'Milk'),
 ('Wine', 'Apple'),
 ('Chips', 'Bread'),
 ('Chips', 'Butter'),
 ('Chips', 'Milk'),
 ('Chips', 'Apple'),
 ('Bread', 'Butter'),
 ('Bread', 'Milk'),
 ('Bread', 'Apple'),
 ('Butter', 'Milk'),
 ('Butter', 'Apple'),
 ('Milk', 'Apple')]
```

Đếm số lần các cặp xuất hiện

```
def count_fre_items(records, itemsets, sup):
    count_item = {}
    for item_set in itemsets:
        for row in records:
            set_A = []
            set_B = []
            for item in item_set:
                set_A.append(item)
            for item in row:
                set_B.append(item)

            if isinstance(set_A, list) and isinstance(set_B, list) and all(elem in set_B for elem in set_A):
                if item_set in count_item.keys():
                    count_item[item_set] += 1
                else:
                    count_item[item_set] = 1

    data = pd.DataFrame()
    data['item_sets'] = count_item.keys()
    data['supp_count'] = count_item.values()
    data = data[data.supp_count >= sup]

    return data
```

```
z=count_fre_items(records, y, len(records) *0.5)
z
```

	item_sets	supp_count
1	(Wine, Bread)	14
2	(Wine, Butter)	12
3	(Wine, Milk)	15
4	(Wine, Apple)	12
9	(Bread, Butter)	14
10	(Bread, Milk)	14
11	(Bread, Apple)	13
12	(Butter, Milk)	14
13	(Butter, Apple)	12
14	(Milk, Apple)	12

Tạo hàm tính confidence và lift

```
def calculate_conf(value1, value2):
    return round(int(value1) / int(value2) * 100, 2)
def calculate_lift(observed_support, antecedent_support, consequent_support, total_transactions):
    if antecedent_support.item() == 0 or consequent_support.item() == 0:
        return None
    expected_support = (antecedent_support.item() * consequent_support.item()) / total_transactions
    lift = observed_support.item() / expected_support
    return lift
```

Trong đó, hàm tính lift gồm:

Observed_support: hỗ trợ quan sát

Antecedent_support: hỗ trợ tiền đề

Consequent_support: hỗ trợ kết quả

Total_transactions: tổng số

Tính các rule cho các items

```
def create_rules(freq_item_sets, threshold):
    rules = []
    for row in freq_item_sets.item_sets:
        if len(row) == 3:  # Chỉ xét nhóm 3 nhân tố
            for i in range(len(row)):
                for j in range(i+1, len(row)):  # Sử dụng i+1 để tránh kiểm tra các cặp trùng lặp
                    for k in range(j+1, len(row)):  # Sử dụng j+1 để tránh kiểm tra các cặp trùng lặp
                        antecedent_support = count_item(records)[count_item(records).item_sets == row[i]].supp_count
                        confidence = calculate_conf(
                            freq_item_sets[freq_item_sets.item_sets == row].supp_count,
                            antecedent_support
                        )
                        if confidence >= threshold:
                            lift = calculate_lift(
                                freq_item_sets[freq_item_sets.item_sets == row].supp_count,
                                antecedent_support,
                                count_item(records)[count_item(records).item_sets == row[j]].supp_count,
                                len(records)
                            )
                            if lift is not None and not np.isnan(lift):
                                rule = {
                                    'items': frozenset([row[i], row[j], row[k]]),
                                    'support': freq_item_sets[freq_item_sets.item_sets == row].supp_count.values[0],
                                    'ordered_statistics': [{
                                        'items_base': frozenset([row[i], row[j]]),
                                        'items_add': frozenset([row[k]]),
                                        'confidence': confidence / 100,
                                        'lift': lift
                                    }]
                                }
                                rules.append(rule)
    return rules
```

Tính ra các giá trị support, confidence, lift

Dựa vào các hàm đã tạo trên. Tạo thuật toán Apriori

```
def apriori(trans_data, minsup=0.5, minconf=0.7):
    freq = pd.DataFrame()
    df = count_item(trans_data)
    sup = len(trans_data) * minsup
    conf = 100 * minconf
    rule = []
    while len(df) != 0:

        itemsets = frequently_items(df.item_sets)
        if itemsets is None:
            return freq, rule

        df = count_fre_items(trans_data, itemsets, sup)
        if len(df) >1 or (len(df) == 3 and int(df.supp_count >= sup)):
            rule = create_rules(df, conf)
            freq = df
            if (len(df) == 3 and int(df.supp_count >= sup)):
                return rule
    return rule
```

Trong vòng lặp while, tìm các items bắt cặp từ 2 liên kết trở lên. Đặt điều kiện if để thực hiện điều đó

```
re_results = apriori(records, minsup=0.5, minconf=0.7)
as_re = list ( re_results)
as_re
```

```
[              item_sets  supp_count
 1    (Wine, Bread, Milk)          12
 6  (Bread, Butter, Milk)          12,
 [{'items': frozenset({'Bread', 'Milk', 'Wine'}),
   'support': 12,
   'ordered_statistics': [{'items_base': frozenset({'Bread', 'Wine'}),
     'items_add': frozenset({'Milk'}),
     'confidence': 0.7059000000000001,
     'lift': 0.9550173010380623}]},
  {'items': frozenset({'Bread', 'Butter', 'Milk'}),
   'support': 12,
   'ordered_statistics': [{'items_base': frozenset({'Bread', 'Butter'}),
     'items_add': frozenset({'Milk'}),
     'confidence': 0.7059000000000001,
     'lift': 1.0147058823529413}]}]]
```

Sử dụng lại hàm in kết quả ở trên ta có được

	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
0	Wine, Bread	Milk	12	0.7059	0.955017
1	Butter, Bread	Milk	12	0.7059	1.014706